CS 3800 Project 1

Auraiporn Auksorn(aauksorn@cpp.edu)

(1) Introduction

The program consists of two part; the server and the client will run on two different machines. The client will take 3 arguments input for IP address, input file, and output file. The client will read input file and write to the server. The server simply listens to the connection from the client, and once the connection establishes the server just sends a file back to the client.

(2) Main Components

Client.java

- Consist of functions to write the args[2] to the server.
- Send file to the server.
- Receive the file from the server.
- Write to a data file.

Server.java.

- Get args[2] from the client.
- Read file from the client.
- Write file back to the client.

GenerateFile.java

- To create input_data_file, I write a program to simply write a letter a to a file according to the given size, and I also generated files by using the online tool.

Calculation.java

- Calculate and record the values of throughput, average RTT, latency, effective bandwidth, and percent of throughput used of the network's total bandwidth.

(3) Challenges

3.1 Challenge #1: I ran into the issue of having the while loop runs into infinite loop when trying to read and write the ObjectOutputStream and the ObjectInputStream.

I later found out that it has to do with the amount of number being declared in the length of byte array. I think the loop condition does not stop since it does know whether it finishes reading yet. For this reason, I came up with the solution of keeping track of number of bytes have been read so far and the amount of the file size.

3.4 Challenge #4: If I understand the project description correctly, I found out that the instructions are not arranged in order. I would have to jump to another instruction when doing

the current instruction. At first, I thought that the transfer time and RTT are the same thing; also, I thought we use average RTT to find the throughput until I carefully observe the formula to find effective bandwidth. Then, I figured out that they are different. According to the lectures, I found out that latency = RTT/2, and effective bandwidth = file size/ latency. According to these formulas would help me to get the transfer time.
TransferTime = RTT + (1/EffectiveBandwidth) * TransferSize

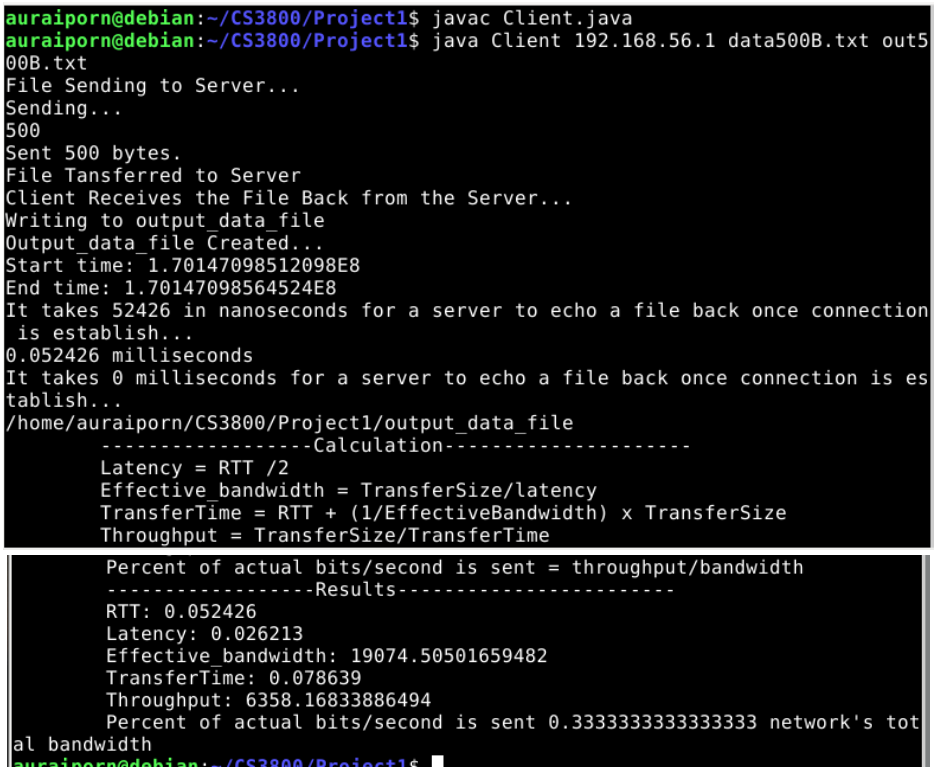
3.5 Challenge #5: This is my first time that I learn computer network programming.

During the lectures, we spent a little bit of time to talk about the coding part besides the lectures itself. For this reason, I have spent a lot of time making a research, reading the articles, watching online tutorial videos, and reading Java documentations. However, I still think that I have learned a lot more details about network programming.

(4) Results

Screenshots of Program Output

Output Client Side:



The file 500B.txt is created:

```
auraiporn@debian:~/CS3800/Project1$ ls -alF
total 9324
drwxr-xr-x 4 auraiporn auraiporn    4096 Sep 29 13:08 ./
drwxr-xr-x 3 auraiporn auraiporn    4096 May  8 13:45 ../
-rw-r--r-- 1 auraiporn auraiporn    4224 Sep 29 13:08 Client.class
-rw-r--r-- 1 auraiporn auraiporn    4277 Sep 29 12:45 Client.java
-rw-r--r-- 1 auraiporn auraiporn   10000 Sep 28 23:49 data10000B.txt
-rw-r--r-- 1 auraiporn auraiporn 2000000 Sep 28 23:40 data2000000B.txt
-rw-r--r-- 1 auraiporn auraiporn  400001 Sep 29 11:13 data400000B.txt
-rw-r--r-- 1 auraiporn auraiporn     500 Sep 28 23:47 data500B.txt
-rw-r--r-- 1 auraiporn auraiporn 7000000 Sep 28 23:44 data7000000B.txt
-rw-r--r-- 1 auraiporn auraiporn   80002 Sep 29 11:25 data80000B.txt
-rw-r--r-- 1 auraiporn auraiporn     731 Sep 29 10:06 GenerateFile.class
-rw-r--r-- 1 auraiporn auraiporn     625 Sep 29 10:06 GenerateFile.java
drwxr-xr-x 2 auraiporn auraiporn    4096 Sep 28 23:16 input_files/
-rw-r--r-- 1 auraiporn auraiporn     500 Sep 29 13:08 out500.txt
drwxr-xr-x 2 auraiporn auraiporn    4096 Sep 29 09:15 source_code/
```

```
auraiporn@debian:~/CS3800/Project1$ cat data500B.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
auraiporn@debian:~/CS3800/Project1$ cat out500.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAauraiporn@debian:~/CS3800/Project1$
```

Output Server Side:

```
C:\Users\acer\eclipse-workspace\Project1_CS3800\src>javac Server.java

C:\Users\acer\eclipse-workspace\Project1_CS3800\src>java Server
Server Listening to Incoming Socket Connection...
Client Connected...
Server Receives a File...
Server is Echoing a File...
Server Done Transferring a File Back to Client...
Socket is Closed Now...
```

Screenshot of all the output files being created on the client machine:

Output files of Second Attempt:

```
-rw-r--r-- 1 auraiporn auraiporn     627 Sep 29 13:28 GenerateFile.java
drwxr-xr-x 2 auraiporn auraiporn    4096 Sep 28 23:16 input_files/
-rw-r--r-- 1 auraiporn auraiporn   10000 Sep 29 13:13 out10000B1.txt
-rw-r--r-- 1 auraiporn auraiporn   10000 Sep 29 13:13 out10000B2.txt
-rw-r--r-- 1 auraiporn auraiporn   10000 Sep 29 13:12 out10000B.txt
-rw-r--r-- 1 auraiporn auraiporn 2000000 Sep 29 13:24 out2MGb1.txt
-rw-r--r-- 1 auraiporn auraiporn 2000000 Sep 29 13:24 out2MGb2.txt
-rw-r--r-- 1 auraiporn auraiporn 2000000 Sep 29 13:23 out2MGb.txt
-rw-r--r-- 1 auraiporn auraiporn 4000000 Sep 29 13:33 out4000000B1.txt
-rw-r--r-- 1 auraiporn auraiporn 4000000 Sep 29 13:34 out4000000B2.txt
-rw-r--r-- 1 auraiporn auraiporn 4000000 Sep 29 13:33 out4000000B.txt
-rw-r--r-- 1 auraiporn auraiporn  400001 Sep 29 13:22 out400000B1.txt
-rw-r--r-- 1 auraiporn auraiporn  400001 Sep 29 13:22 out400000B2.txt
-rw-r--r-- 1 auraiporn auraiporn  400001 Sep 29 13:23 out400000B4.txt
-rw-r--r-- 1 auraiporn auraiporn  400001 Sep 29 13:21 out400000B.txt
-rw-r--r-- 1 auraiporn auraiporn     500 Sep 29 13:11 out500B1.txt
-rw-r--r-- 1 auraiporn auraiporn     500 Sep 29 13:11 out500B2.txt
-rw-r--r-- 1 auraiporn auraiporn     500 Sep 29 13:08 out500.txt
-rw-r--r-- 1 auraiporn auraiporn   80002 Sep 29 13:15 out80000B1.txt
-rw-r--r-- 1 auraiporn auraiporn   80002 Sep 29 13:15 out80000B2.txt
-rw-r--r-- 1 auraiporn auraiporn   80002 Sep 29 13:15 out80000B3.txt
-rw-r--r-- 1 auraiporn auraiporn   80002 Sep 29 13:14 out80000B.txt
drwxr-xr-x 2 auraiporn auraiporn    4096 Sep 29 09:15 source_code/
```

Output files of Last Attempt:

```
-rw-r--r-- 1 auraiporn auraiporn 4000000 Sep 29 13:28 data4000000B.txt
-rw-r--r-- 1 auraiporn auraiporn  400000 Sep 29 15:37 data400000B.txt
-rw-r--r-- 1 auraiporn auraiporn     500 Sep 28 23:47 data500B.txt
-rw-r--r-- 1 auraiporn auraiporn   80000 Sep 29 15:45 data80000B.txt
-rw-r--r-- 1 auraiporn auraiporn     732 Sep 29 15:45 GenerateFile.class
-rw-r--r-- 1 auraiporn auraiporn     624 Sep 29 15:44 GenerateFile.java
drwxr-xr-x 2 auraiporn auraiporn    4096 Sep 28 23:16 input_files/
-rw-r--r-- 1 auraiporn auraiporn   10000 Sep 29 15:40 out10000B1.txt
-rw-r--r-- 1 auraiporn auraiporn   10000 Sep 29 15:40 out10000B2.txt
-rw-r--r-- 1 auraiporn auraiporn   10000 Sep 29 15:40 out10000B.txt
-rw-r--r-- 1 auraiporn auraiporn 2000000 Sep 29 15:50 out2MGB1.txt
-rw-r--r-- 1 auraiporn auraiporn 2000000 Sep 29 15:51 out2MGB2.txt
-rw-r--r-- 1 auraiporn auraiporn 2000000 Sep 29 15:50 out2MGB.txt
-rw-r--r-- 1 auraiporn auraiporn 4000000 Sep 29 15:52 out4MGB1.txt
-rw-r--r-- 1 auraiporn auraiporn 4000000 Sep 29 15:53 out4MGB2.txt
-rw-r--r-- 1 auraiporn auraiporn 4000000 Sep 29 15:52 out4MGB.txt
-rw-r--r-- 1 auraiporn auraiporn     500 Sep 29 15:33 out500B1.txt
-rw-r--r-- 1 auraiporn auraiporn     500 Sep 29 15:33 out500B2.txt
-rw-r--r-- 1 auraiporn auraiporn     500 Sep 29 15:29 out500B.txt
-rw-r--r-- 1 auraiporn auraiporn   80000 Sep 29 15:48 out80000B1.txt
-rw-r--r-- 1 auraiporn auraiporn   80000 Sep 29 15:49 out80000B2.txt
-rw-r--r-- 1 auraiporn auraiporn   80000 Sep 29 15:48 out80000B.txt
drwxr-xr-x 2 auraiporn auraiporn    4096 Sep 29 09:15 source_code/
auraiporn@debian:~/CS3800/Project1$
```

RTT Values of the Second Attempt

<u>File Size: 500 bytes</u>

RTT1 = 0.057956 milliseconds, RTT2 = 0.108325 milliseconds, RTT3 = 0.094773 milliseconds

Average RTT of file size: 500 bytes is 0.087018.

<u>File Size: 10000 bytes</u>

RTT1 = 0.199424 milliseconds, RTT2 = 0.161206 milliseconds, RTT3 = 0.168834 milliseconds

Average RTT of file size: 10000 bytes is 0.176488.

File Size: 80000 bytes
RTT1 = 1.205587 milliseconds, RTT2 = 1.427295 milliseconds, RTT3 = 1.839852 milliseconds
Average RTT of File Size: 80000 bytes is 1.4909113333333333.
File Size: 400000 bytes
RTT1 = 15.403034 milliseconds, RTT2 = 16.026232 milliseconds, RTT3 = 18.805656 milliseconds
Average RTT of File Size: 400000 bytes is 16.744974.
File Size: 2000000 bytes
RTT1 = 75.97204 milliseconds, RTT2 = 73.271662 milliseconds, RTT3 = 81.007002 milliseconds
Average RTT of File Size: 2000000 bytes is 76.75023466666667.
File Size: 4000000 bytes
RTT1 = 164.6329 milliseconds, RTT2 = 175.268644 milliseconds, RTT3 = 187.499148 milliseconds
Average RTT of File Size: 4000000 bytes is 175.80023066666664.

RTT Values of the Last Attempt

File Size: 500 bytes
RTT1 = 0.052426 milliseconds, RTT2 = 0.104331 milliseconds, RTT3 = 0.090941 milliseconds
Average RTT of File Size: 500 is: 0.08256599999999999
File Size: 10000 bytes
RTT1 = 0.135148 milliseconds, RTT2 = 0.147655 milliseconds, RTT3 = 0.19225 milliseconds
Average RTT of File Size: 10000 is: 0.15835100000000002
File Size: 80000 bytes
RTT1 = 6.02916 milliseconds, RTT2 = 1.622773 milliseconds, RTT3 = 3.170046 milliseconds
Average RTT of File Size: 80000 is: 3.607326333333333
File Size: 400000 bytes
RTT1 = 18.505549 milliseconds, RTT2 = 18.317537 milliseconds, RTT3 = 16.538636 milliseconds
Average RTT of File Size: 400000 is: 17.787240666666666
File Size: 2000000 bytes
RTT1 = 75.437004 milliseconds, RTT2 = 83.860507 milliseconds, RTT3 = 79.140108 milliseconds
Average RTT of File Size: 2000000 is: 79.47920633333332
File Size: 4000000 bytes
RTT1 = 153.016 milliseconds, RTT2 = 159.874904 milliseconds, RTT3 = 157.330776 milliseconds
Average RTT of File Size: 4000000 is: 156.74056

Calculation

*All the screenshots are the values from the second attempt, and the detail about last attempt can be found on Calculation.java

- In this experiment, RTT = latency + data size/ bandwidth.

```
------------------Latency---------------------
Latency of file size 500 is: 0.043509
Latency of file size 10000 is: 0.088244
Latency of file size 80002 is: 0.7454556666666666
Latency of file size 400001 is: 8.372487
Latency of file size 2000000 is: 38.375117333333336
Latency of file size 4000000 is: 87.90011533333332
```

- As the file gets bigger, the latency is also increased because latency is half of the round-trip time.

```
------------------EffectiveBandwidth--------------------
Effective Bandwidth of file size 500 is: 11491.875244202349
Effective Bandwidth of file size 10000 is: 113322.15221431485
Effective Bandwidth of file size 80002 is: 107319.59468191044
Effective Bandwidth of file size 400001 is: 47775.648979807316
Effective Bandwidth of file size 2000000 is: 52117.104493196246
Effective Bandwidth of file size 4000000 is: 45506.19740180395
```

- According to the lecture states “[b]andwidth is the number of bits per second that can be sent through a link.” I memorize bandwidth as bits/ second which is the maximum possible transfer rate. In this experiment, I find that the larger file size is sent via socket; the smaller number of bits per second can be sent out through socket.

```
------------------Average RTT---------------------
Average RTT of File Size: 500 is: 0.087018
Average RTT of File Size: 10000 is: 0.176488
Average RTT of File Size: 80002 is: 1.4909113333333333
Average RTT of File Size: 400001 is: 16.744974
Average RTT of File Size: 2000000 is: 76.75023466666667
Average RTT of File Size: 4000000 is: 175.80023066666664
```

```
------------------TransferTime---------------------
Transfer Time of file size 500 is: 0.130527
Transfer Time of file size 10000 is: 0.264732
Transfer Time of file size 80002 is: 2.236367
Transfer Time of file size 400001 is: 25.117461
Transfer Time of file size 2000000 is: 115.125352
Transfer Time of file size 4000000 is: 263.70034599999997
```

- Also, I think that RTT and transfer time keep increasing when sending larger files because the effective bandwidth decreases, and a smaller number of bits per second can be sent through a link.
- In order to measure the network performance, I think we would have to observe at latency and effective bandwidth. I think speed plays an important role of evaluating the network performance as well because the delay or latency that occurs would affect the speed. When sending a small amount of data, we should expect it to be fast. I think measuring the network performance when sending a larger file, we would have to observe the effective bandwidth of a network. When sending a large amount of data, we

should expect the ability of the network to be able to send larger number of bits per second through a link. We would have to consider the maximum capacity of number of bits per second can be sent.

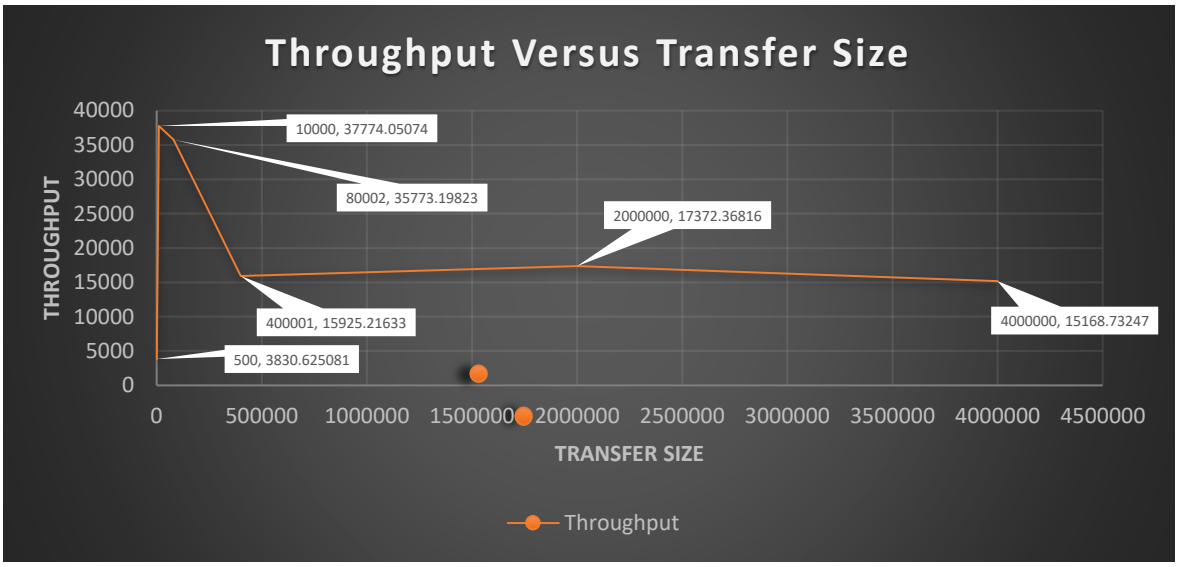
```
------------------Throughput---------------------
Throughput of file size 500 is: 3830.625081400783
Throughput of file size 10000 is: 37774.05073810495
Throughput of file size 80002 is: 35773.19822730348
Throughput of file size 400001 is: 15925.216326602438
Throughput of file size 2000000 is: 17372.36816439875
Throughput of file size 4000000 is: 15168.732467267982

------------------Percent of Throughput Used of the Network's Total Bandwidth---------------------
The throughput 500 is 0.33333333333333337 of the network's total bandwidth.
The throughput 10000 is 0.3333333333333333 of the network's total bandwidth.
The throughput 80002 is 0.3333333333333333 of the network's total bandwidth.
The throughput 400001 is 0.3333333333333333 of the network's total bandwidth.
The throughput 2000000 is 0.33333333333333337 of the network's total bandwidth.
The throughput 4000000 is 0.3333333333333333 of the network's total bandwidth.
```

- According to the lecture, it states “[t]hroughput is the actual transfer rate archived.” In this experiment, I notice that the results of throughput of different files are slightly different. I think there are not any differences of the throughput, so I calculated the percent of number of bits per second used of the network’s total bandwidth. I find that the throughput was 0.33% of the network’s total bandwidth when data size is in the range of 500B – 4000000B. I would conclude that they have the same throughput.

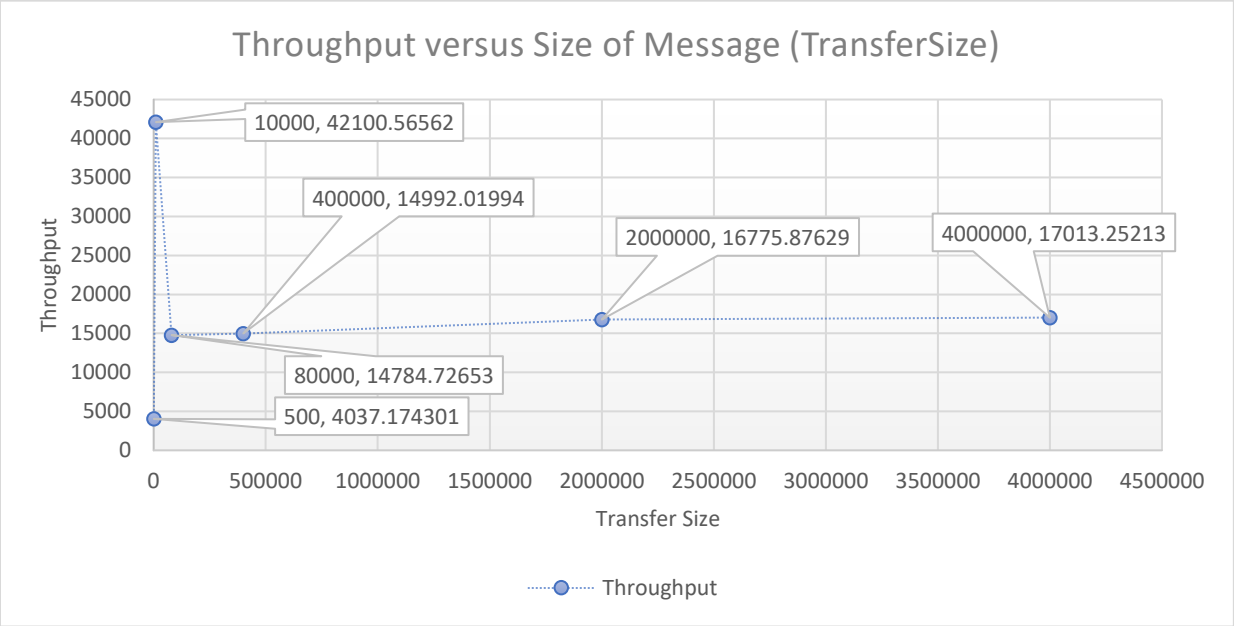
A graph with Throughput versus Size of Message (TransferSize)

This is the graph of the second attempt:



Transfer Size	Throughput
500	3830.625081
10000	37774.05074
80002	35773.19823
400001	15925.21633
2000000	17372.36816
4000000	15168.73246

This is the graph of the last attempt:

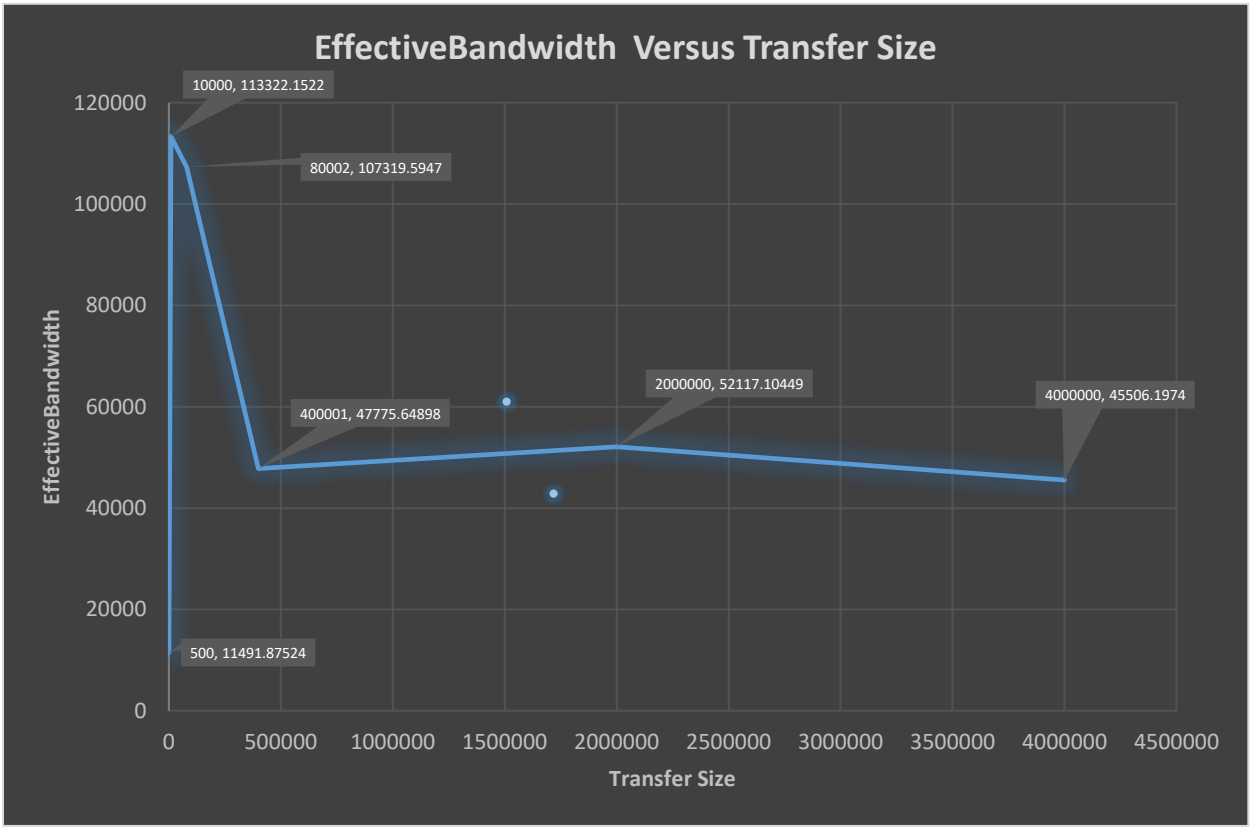


Transfer Size	Throughput
500	4037.174301
10000	42100.56562
80000	14784.72653
400000	14992.01994
2000000	16775.87629
4000000	17013.25213

According to the observations of the graph, I am not sure why the throughputs of the transfer size of the range of 500B-80000B are not stable. It could be the noises that could interfere with the connections. However, a graph throughput versus transfer size shows the increasing line of throughput as the file size gets larger. Reading a larger file would have to send higher number of bits per second through a socket. Although the throughputs of different file sizes are slightly different, they only use 0.33% out of the network total's bandwidth. In other words, as the files are larger, it would reduce the maximum of possible transfer rate which has the unit in bites per second. I also attempted to record RTT values to crate the graph twice, and the result comes out to be the same.

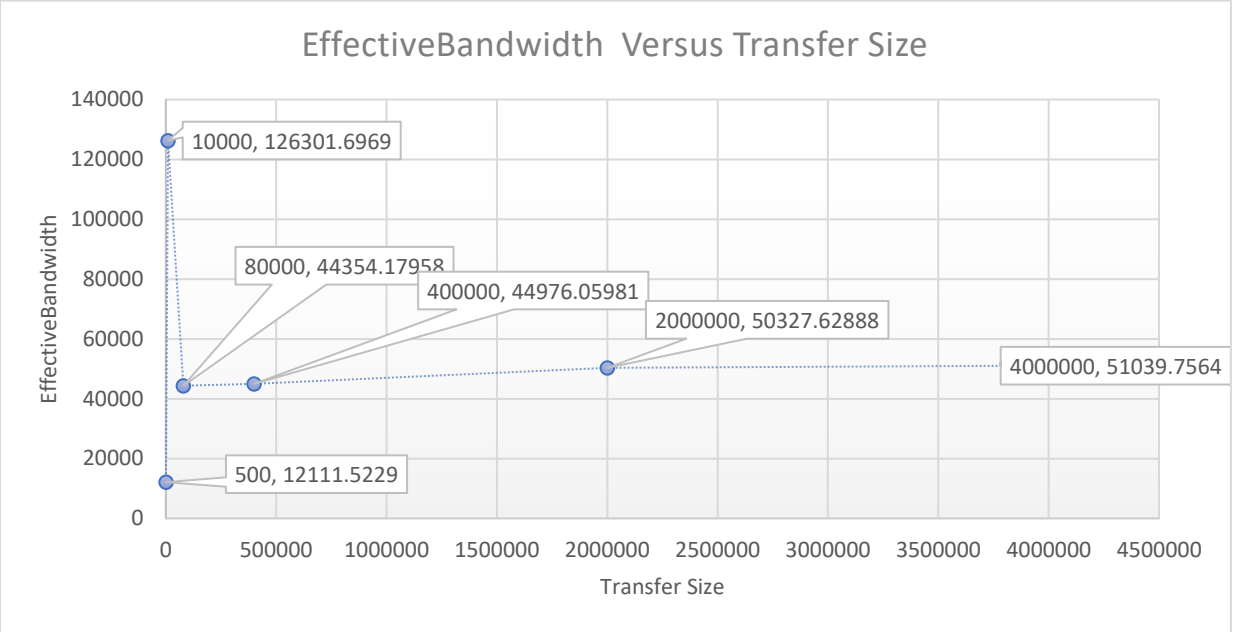
A graph with Effective Bandwidth (EffectiveBandwidth) versus Size of Message (TransferSize)

This is the graph of the second attempt:



Transfer Size	EffectiveBandwidth
500	11491.87524
10000	113322.1522
80002	107319.5947
400001	47775.64898
2000000	52117.10449
4000000	45506.1974

This is the graph of the last attempt:



Transfer Size	EffectiveBandwidth
500	12111.5229
10000	126301.6969
80000	44354.17958
400000	44976.05981
2000000	50327.62888
4000000	51039.7564

According to the graph, I notice that the effective bandwidth is decreasing as the file gets larger. It means that when sending or receiving a larger file, the maximum possible transfer rate will decline.

(5) Evaluation

With no background in computer programming, I strived for when I first started working on the first project and optimizing the code by understanding the functionality of the socket to ensure that this project would meet the requirement. I did my best to try to understand the topics that we have been discussed in class and used the knowledge from our class to apply with the first project. I think I have learned about network programming more while accomplishing this project.